Claims

1. A method for preprocessing Java application code, comprising the operations of:

receiving a Java template file, the Java template file including Java programming language code and meta code;

processing the Java template to create an intermediate program using the meta code, wherein the intermediate program is a Java program;

compiling the intermediate program to create an intermediate class, wherein the intermediate class is a Java based class; and

generating an object text file using the intermediate class.

2. A method as recited in claim 1, wherein the meta code can include Java programming language statements.

3. A method as recited in claim 2, wherein the meta code is equivalent to the Java programming language.

4. A method as recited in claim 1, further comprising the operation of transforming lines in the Java template.

5. A method as recited in claim 4, wherein the transformed lines are entered into the intermediate program.

6. A method as recited in claim 5, further comprising the operation of entering header data into the intermediate program.

7. A method as recited in claim 5, further comprising the operation of entering footer data into the intermediate program.

8. A method for transforming lines of a Java template for preprocessing, comprising the operations of:

obtaining a line of text from a Java template;

determining if the line of text begins with a meta symbol, wherein the meta symbol is a predefined symbol indicating the line of text is a preprocessor directive;

removing the meta symbol from the line of text when the line of text begins with the meta symbol; and

transforming the line of text into a function argument when the line of text does not begin with the meta symbol.

9. A method as recited in claim 8, wherein transforming the line of text into a function argument comprises the operation of enclosing the line of text in quotes.

10. A method as recited in claim 9, wherein transforming the line of text into a function argument further comprises the operation of prepending the line of text with an internal preprocessor method call and an open bracket.

11. A method as recited in claim 10, wherein transforming the line of text into a function argument further comprises the operation of appending a closing bracket and a semicolon to the end of the line of text.

12. A method as recited in claim 11, wherein the quotes are double quotes.

13. A method as recited in claim 8, further comprising the operation of determining whether macros are present in the line of text.

14. A method as recited in claim 13, further comprising the operation of transforming macros present in the line of text into appropriate instructions.

15. A method as recited in claim 8, further comprising the operation of writing header data, footer data, and the transformed lines of text to an intermediate program.

16. A Java preprocessor, comprising:

a meta code converter capable of processing a Java template to create an intermediate program using meta code included in the Java template; and

an object text generator that compiles the intermediate program to create an intermediate class, wherein the intermediate class is a Java based class, the object text generator also capable of generating an object text filed using the intermediate class.

17. A Java preprocessor as recited in claim 16, wherein the meta code can include Java programming language statements.

18. A Java preprocessor as recited in claim 17, wherein the meta code is equivalent to the Java programming language.

19. A Java preprocess as recited in claim 18, further comprising a preprocessor library interface module that specifies a contract that can be extended by any Java preprocessor library that is to be supported by the Java preprocessor.

20. A Java preprocessor as recited in claim 19, further comprising an executor module that calculates the name of the object text generator class and invokes main method of the object text generator class.